Numerical Simulations For Active Tectonic Processes: Increasing Interoperability And Performance

JPL Task Plan No. 83-6791

Interoperability Milestone H – 7/30/2002

Come to agreement on design policy for interoperability and community delivery - Review board approves requirements and a preliminary design for functionality.

Team

Geoffrey Fox: Information Architect

Community Grid Computing laboratory
Indiana University
501 N. Morton, Suite 224
Bloomington, IN 47404-3730
gcf@indiana.edu
812-856-7977

Jay Parker: Overall software engineer

Jet Propulsion Laboratory
Mail Stop 238-600
4800 Oak Grove Drive
Pasadena, CA 91109-8099
Jay.W.Parker@jpl.nasa.gov
818-354-6790

Andrea Donnellan: Database design and implementation

Jet Propulsion Laboratory
Mail Stop 183-335
4800 Oak Grove Drive
Pasadena, CA 91109-8099
donnellan@jpl.nasa.gov
818-354-4737

Marlon Pierce – Indiana University: Code Interoperability Software Engineer

Community Grid Computing laboratory
Indiana University
501 N. Morton, Suite 224
Bloomington, IN 47404-3730
marpierc@indiana.edu
812-856-1212

Dennis McLeod: Database interoperability

University of Southern California
Mail Code 0781
3651 Trousdale Parkway
Los Angeles, CA 90089-0742
mcloed@pollux.usc.edu
213-740-7285

Anne Chen: Database implementation

University of Southern California
Mail Code 0781
3651 Trousdale Parkway
Los Angeles, CA 90089-0742
yunanche@usc.edu
213-740-7285

Lisa Grant: Fault database architect

University of California, Irvine
Environmental Analysis and Design
Irvine, CA 92697-7070
lgrant@uci.edu
949-824-5491

Miryha Gould: Population of fault database

University of California, Irvine
Environmental Analysis and Design
Irvine, CA 92697-7070
miryha@uci.edu
949-824-5491

Application

We are building a new Problem Solving Environment (QuakeSim) for use by the seismological, crustal deformation, and tectonics communities for developing an understanding of active tectonic and earthquake processes. The top-level operational architecture of our proposed solid earth research virtual observatory (SERVO) shows science users interacting with interface programs as well as modeling, simulation, and analysis tools. The general architecture follows the "Web Services" model being developed by business interests, but is applied to scientific applications and supporting software resources (such as databases). The system is divided into three tiers: a user interface layer (implemented as a browser interface), a system resource layer, and a middle control layer that maintains proxies (or brokers) to the system resources (Figure 1). The middle tier provides a uniform interface to the resource layer. Following the Web Services approach, we define XML interface abstractions (in WSDL) for basic services (such as File Management) and implement the interface with appropriate technologies (such as with a relational database). Communication between the services is done with an XML messaging architecture (SOAP).

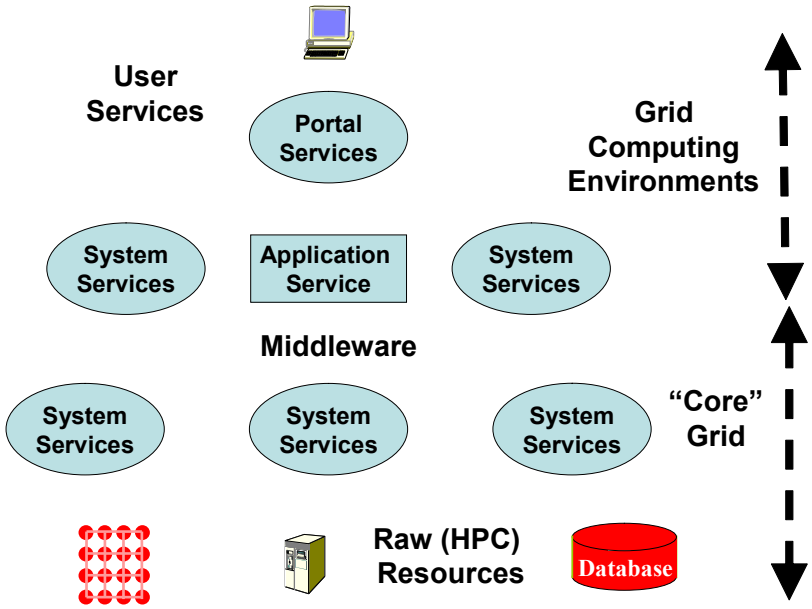

Figure 1. High level architecture of planned system showing grids, portals, and grid computing environments.

One of the most critical aspects of our proposed system is supporting interoperability given the heterogeneous nature of data sources as well as the variety of application programs, tools, and simulation packages that must operate with data from our system. Interoperability will be implemented by using distributed object technology combined with development of object API's that conform to emerging standards. We will define our object API's in XML and dynamically map this specification into the chosen object model. This strategy was successfully used in the Gateway portal, which currently uses a CORBA middle tier but has used a pure Java solution with the same objects.

Our objective is to develop a system with the following specific components.

1. A database system for handling both real and simulated data.
2. Fully three-dimensional finite element code with adaptive mesh generator capable of running on workstations and supercomputers for carrying out earthquake simulations.
3. Inversion algorithms and assimilation codes for constraining the models and simulations with data.
4. A collaborative portal (Object Grid Framework) allowing for seamless communication between codes, reference models, and data.
5. Pattern recognizers capable of running on workstations and supercomputers for analyzing data and simulations.

For our development we will follow the software engineering plan but may add adaptations in cases where particular unique requirements emerge.

Requirements

The project software engineering plan is located at:

http://www-aig.jpl.nasa.gov/public/dus/gem/CT/SW_Eng_plan.html

Requirements for the project have been defined and can be found at:

http://www-aig.jpl.nasa.gov/public/dus/quakesim/CT_Requirements.doc

http://www-aig.jpl.nasa.gov/public/dus/quakesim/CT_Requirements.pdf

Code descriptions and input/outputs:

Several codes are being implemented in this system for individual use, or for interaction with other codes in the system. The general code descriptions are below as well as a diagram indicating linkages between the codes (Figure 2). Abstractions of each module are at the end of the document (Figures 5-10). The inputs and outputs are specified in terms of how the codes will communicate and will be interoperable. These abstractions do not serve to describe the details of the actual (internal) input and output formats. Details of the codes and documentation are posted separately and links are provided in cases where this has been posted.

Links to posted code and documentation can be found at:

http://www-aig.jpl.nasa.gov/public/dus/quakesim/documentation.html

http://www-aig.jpl.nasa.gov/public/dus/quakesim/download.html

The major project benchmarked codes have been documented and posted. Additional codes throughout the project will be posted and are noted below. Brief descriptions are as follows:

disloc

Handles multiple arbitrarily dipping dislocations (faults) in an elastic half-space to produce surface displacements based on Okada's 1985 paper.

Status: To be posted.

simplex

Inverts surface geodetic displacements for fault parameters using simulated annealing downhill residual minimization. Is based on disloc and uses dislocations in an elastic half space.

Status: To be posted.

geofest (coupled with a mesh generator)

Three-dimensional viscoelastic finite element model for calculating nodal displacements and tractions. Allows for realistic fault geometry and characteristics, material properties, and body forces.

Status: Posted

VC (VirtualCalifornia)

Program to simulate interactions between vertical strike-slip faults using an elastic layer over a viscoelastic half-space.

Status: Posted

park

Boundary element program to calculate fault slip velocity history based on fault frictional properties.

Status: Posted

DAHMM

Time series analysis program based on Hidden Markov Modeling. Produces feature vectors and probabilities for transitioning from one class to another.

Status: To be posted

PDPC

Time series analysis pattern recognition program to calculate anomalous seismicity patterns and probabilities.

Status: To be posted

Flow between code modules

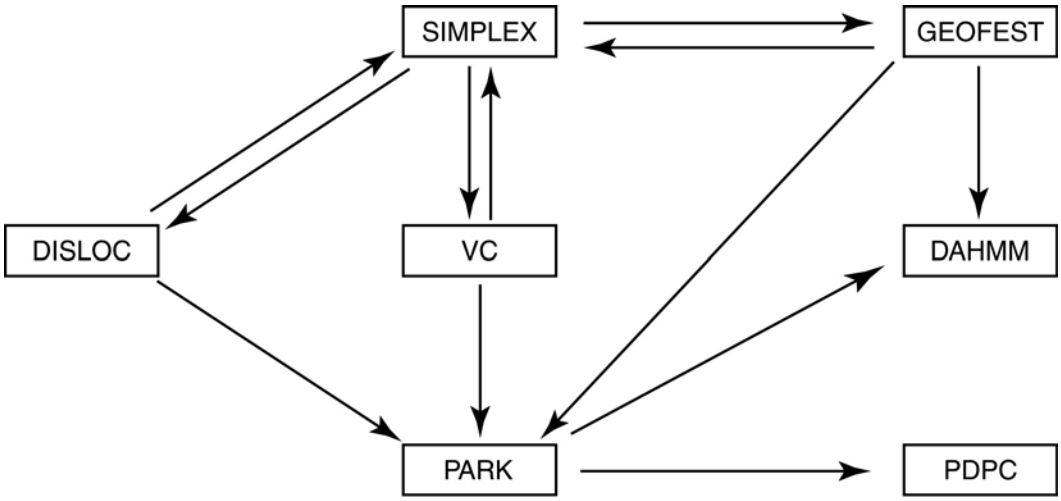

Figure 2. Linkages between programs that maybe run separately or coupled to other programs.

Design Policy

The goal of this project is to provide codes and data in an integrated web-services based environment using a proxy component architecture (Figure 3). To accomplish this goal we must take advantage of rapidly evolving web-services technology. Therefore, it is difficult to specify the design in great detail at the outset of the project. Our work will rely on rapid prototyping, testing, and then deployment. The interfaces that will be developed will come out of research and experimentation. As the team determines the best protocols and methods they will be posted on the web.

Documents posted so far are:

Approach and schema:

http://www.servogrid.org/slide/GEM/Interop/AWS.doc.

Schema definitions:

http://www.servogrid.org/GCWS/Schema/index.html.

WSDL Descriptions of the core service definitions:

http://www.servogrid.org/GCWS/Schema/index.html

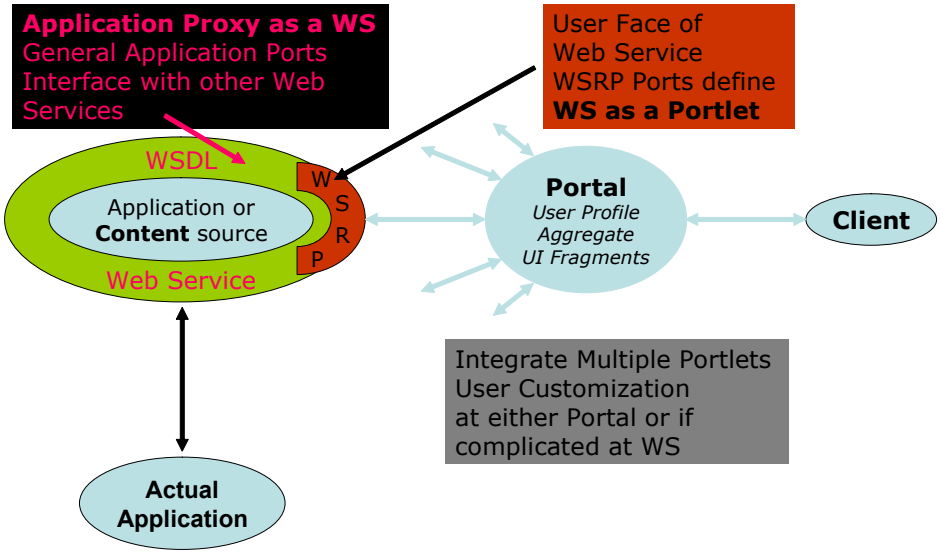

Figure 3 An overview of the proxy component architecture. A complete description can be found at: http://www.servogrid.org/slide/GEM/Interop/AWS.doc.

Interoperability

In our design policy each program becomes a Web service and all data is fed from Web services. This make take place through: 1) another Web service (program), 2) file access implemented as a Web service, or 3) database access wrapped as a Web service. Each data I/O becomes a port with an associated WSDL (Web Services Definition Language to become OGSA Open Grid Service Architecture). For our schema WSDL specifies the data structure of this stream. Other important ports on each Web service are: 1) 'user-facing ports" for Web input and output, and 2) "Grid Information port" where one can enquire as to properties of Web service. The initial code linkages (Figure 1) will be encoded in GSFL Grid Service Flow Language being developed by another group at Indiana and Argonne.

Community Delivery

Community delivery will be via the world wide web. We will make use of the Open Channel Foundation to post the software from this project. We have also established a web site http://www.servogrid.org for disseminating information about the project, documentation, and links to software source code. SERVO is the Solid Earth Research Virtual Observatory, which makes use of grid technologies.

Preliminary Design for Functionality

Fault Database

The "database system" for this project must manage a variety of types of earthquake science data and information. There are pre-existing collections, with heterogeneous access interfaces; there are also some structured collections managed by general-purpose database management systems. We will also construct at least one new database, characterizing dynamically-defined earthquake faults.

We will develop an XML scheme to describe various parameters of earthquake faults and input data. We will develop our earthquake fault databases based on this previous work. The databases will focus on paleoseismic, GPS, InSAR, and seismicity data. We will also work with communities that have begun to establish data standards. Our database will include simulated faults, well-known faults, as well as hypothesized faults, for testing through our simulations. We will tag our data based on the type of interpretation and also indicate the level of confidence that the fault or segment actually exists.

Most faults in the existing databases have been divided into characteristic segments that are proposed to rupture as a unit. Geologic slip rates are assigned to large segments rather than to the specific locations (i.e. geographic coordinates) where they were measured. These simplifications and assumptions are desirable for seismic hazard analysis, but they introduce a level of geologic interpretation and subjective bias that is inappropriate for simulations of fault behavior. Therefore, we propose to develop an objective database that includes primary geologic and paleoseismic fault parameters (fault location/geometry, slip rate at measured location, measurements of coseismic displacement, dates and locations of previous ruptures) as well as separate interpreted/subjective fault parameters such as characteristic segments, average recurrence interval, magnitude of characteristic ruptures etc (Figure 4). Both will be updated as more data is acquired and interpreted through research and the numerical simulations.

To support this earthquake fault database and others, we will eventually acquire and employ a state-of-the-art commercially-available general-purpose database management system (DBMS). Initially our database system will be developed on a public domain DBMS such as MySQL. In particular, we will utilize an extensible relational system. These systems support the definition, storage, access, and control of collections of structured data. Further, we require extensible type definition capabilities in the DBMS (to accommodate application-specific kinds of data), the ability to combine information from multiple databases, and mechanisms to efficiently return XML results from requests. Currently, DBMSs available from Informix, Oracle, and Sybase would provide a good portion of the necessary capabilities.

One key issue that we must address here is the fact that such DBMSs operate on SQL requests, rather than those in some XML-based query language. Query languages for XML are just now emerging; in consequence, we shall initially do XML to SQL translations in our middleware/broker. With time, as XML query language(s) emerge, we will employ them in our system. To provide for the access and manipulation of heterogeneous data sources (datasets, databases), the integration of information from such

sources, and the structural organization and data mining of this data, we propose to devise and employ techniques being developed at the USC Integrated Media Systems Center for wrapper-based information fusion to support data source access and integration [MNN99; AM99].

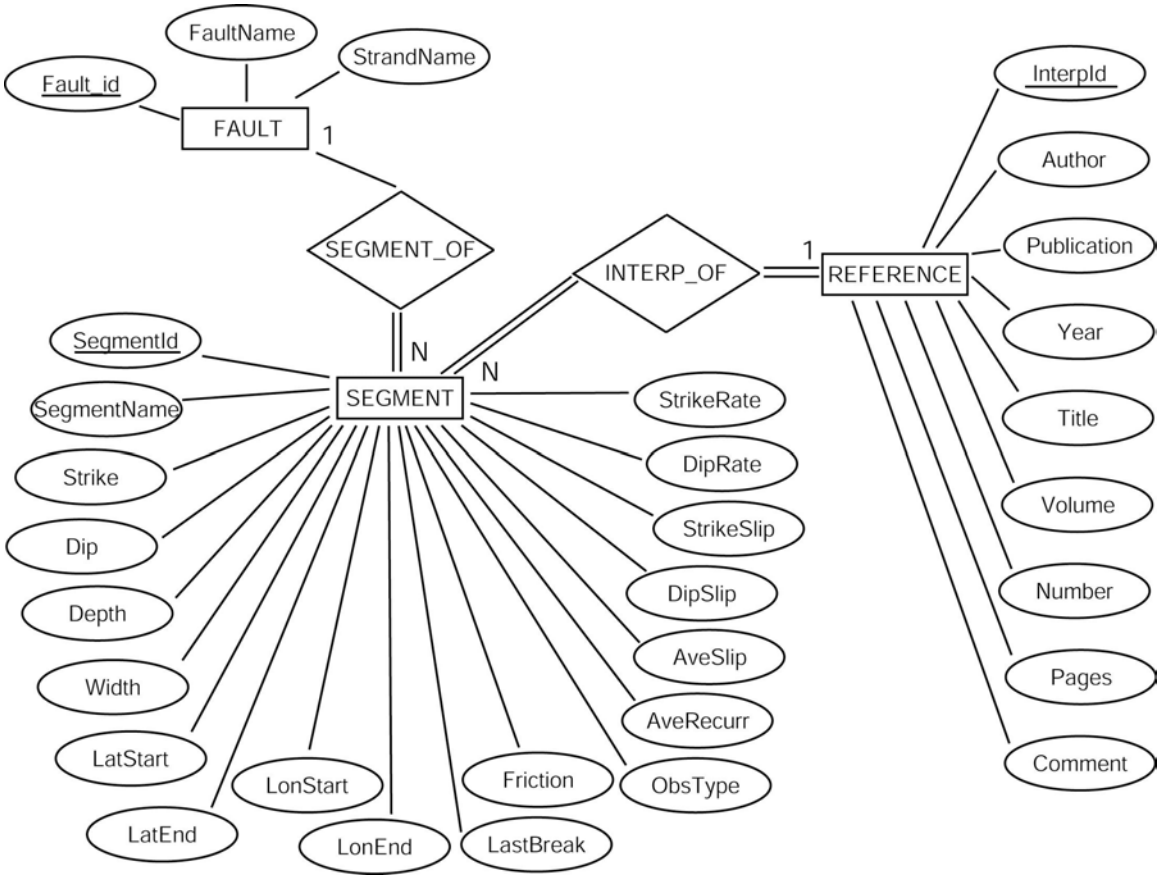

Figure 4. Fault database extended entity relationship. This will be mapped to a relational database and will be initially implemented in MySQL.

Code Interoperability

We are addressing several aspects of code interoperability, including

1. The execution of applications (such as disloc, geofest, etc.) on different resources;
2. Method interfaces to core services such as job submission and file transfer;
3. Common metadata about application input and output data; and
4. Common events that will allow communication between applications.

We do this through XML schemas definitions. As we describe below, applications are defined by application descriptor schema, method interfaces use WSDL (Web Service

Definition Language, a World Wide Web Consortium recommendation). Application data types and events are also being abstracted into XML definitions.

Our approach treats GEM codes as "legacy" applications that do not need to be altered. Instead of building interoperable libraries for the codes, we wrap the codes in general purpose application proxies, which derive from the same set of schema definitions. The application proxies have two primary purposes: they describe the necessary steps to invoke, monitor, and communicate with the application, and they describe the underlying Core Web Services, described below, that are needed by the application.

Core Web Services are basic functions needed to interact with a particular host resource (a workstation, a supercomputer, a database, etc.) We identify the following minimal basic service abstractions: file manipulation, command execution, batch script generation, job monitoring. Each of these services is defined by an interface written in the Web Services Definition Language (WSDL), an XML language for describing class interfaces. This interface can be cast into the appropriate language (such as Java) and implemented by programming to the interface. The advantage of this approach (as we have explored in other projects) is that it allows geographically distributed development groups to collectively define the relatively lightweight interface and then develop the service implementations separately. Client and server implementations will interoperate because a common interface is used and a constrained set of wire protocol communications carry the method invocation requests and responses (typically in SOAP, such as in our prototype systems). Implementation issues are hidden from outside developers by encapsulation. An example of the usefulness of this development technique is exemplified by the file management service, which may rely on a number of technologies (from a simple approach using the Unix file system to various database technologies) in various implementations, but allows for interoperability through a single standard interface.

We have defined the WSDL interfaces for the minimal list of basic services mentioned above and have prototype implementations in place on a testbed PSE. Other core services that can be implemented include basic information services about host computing resources and authorization services that control the access to content.

The core services themselves may be interesting to end users, but more generally we will want to concentrate on applications, describing how a user runs the Geofest application on a particular resource. We have defined a set of schema that we call application descriptors for this purpose. The application descriptor schema has been developed to be an abstract way of describing how to use applications. A particular code has only one application description that can be used by any other component of the system. This description serves as a common data model for all applications, and the associated "get" and "set" methods for manipulating the data serves as a rudimentary programming interface, with more convenient interfaces created through adapters to these rudimentary methods.

We have designed the application description schema for multiple purposes. From one point of view they provide a simple and standard way for adding applications to our problem solving environment. With their associated programming interface, they also define a standard around which client interfaces to the applications can be built. From

the code interoperability point of view, the application descriptions and their programming interfaces define application proxies. These proxies eliminate the need to develop interoperable communication mechanisms within the codes themselves. Instead, in our architecture all communication between applications will go through the application proxy interfaces, so we need to define and develop communication mechanisms only for the proxies. The proxies in turn are responsible for translating incoming and outgoing messages as needed by the applications that they wrap.

One simple but important example of application communication is the posting and receiving of execution status. In support of this, we have developed a simple event mechanism based on queuing system email. An application proxy may be created to run a code on a high performance computer using a queuing system such as PBS. The queuing system generates email at various stages of execution (queued, running, completed), and we treat these as simple events that can be used to communicate between listeners. The queue system email message is captured and converted into a generic XML format and posted to a publish/subscribe system (such as the Java Message Service in our implementation), which then publishes the event to all subscribers. Subscribing applications may include job status displays, email forwarders that resend the notice to the user, and other applications proxies that need to know the status of the running application. We intend for this simple event system to form the basis of our prototype code coupling system: applications can be sequentially staged, with the next application in a chain executing after it receives the "completed" event from the previous application.

Finally, there is the problem of data interoperability: different codes need to work with each others data in a data flow chain. Our plan for interoperable data descriptions is to define the general metadata tags that describe the content of our target codes. We show these metadata abstractions for each of our targeted codes graphically in the figures. We can then use this as a common data format and provide translators between the common format and the particular format needed by a given code. This eliminates the need to build a set of translators between every code pair. There are actually several existing "common data formats" that may be appropriate, such as HDF5 and NetCDF, as well as custom hybrid approaches such as the XDMF from the DoD's ICE project.

Documented technical achievements towards goal

In addition to defining the design specifications outlined here, we have begun the initial development of the system. A prototype database using MySQL with a java interface has been implemented and tested. The module disloc has been put into the Gateway framework and runs as a web service.

- We have designed WSDL (XML) interfaces for basic portal services: job submission, batch script generation, file transfer, and job monitoring. The preliminary WSDL descriptions of the core service definitions are available from http://www.servogrid.org/GCWS/servlet/AxisServlet. These service definitions are subject to change, and updated service definitions will appear at the above URL automatically when the new versions of any particular service are deployed. All of the schema definitions included in that document are available at: http://www.servogrid.org/GCWS/Schema/index.html.

- We have made initial implementations of clients and servers for the above services. Clients and servers are implemented in Java, and communication is handled using SOAP over HTTP requests and responses.
- We have designed initial sets of schema for describing applications and have implemented Java language bindings for these schema as well as interfaces for manipulating the data objects. Our schema and our approach are documented in a report available at http://www.servogrid.org/slide/GEM/Interop/AWS.doc.
- We have developed a prototype portal interface for the disloc code using the above services and application schema. An advanced version of the portal also supports simplex and allows execution and management of jobs and files on three different computing resources (a linux box running PBS, a solaris workstation that runs jobs interactively, and a 64 processor Sun E10000 running PBS).
- We have set up a project web server, http://www.servogrid.org/ that includes support for distributed content management, allowing individual group members to upload files, documents, and codes, and to manage their own web content.

Abstractions of code modules

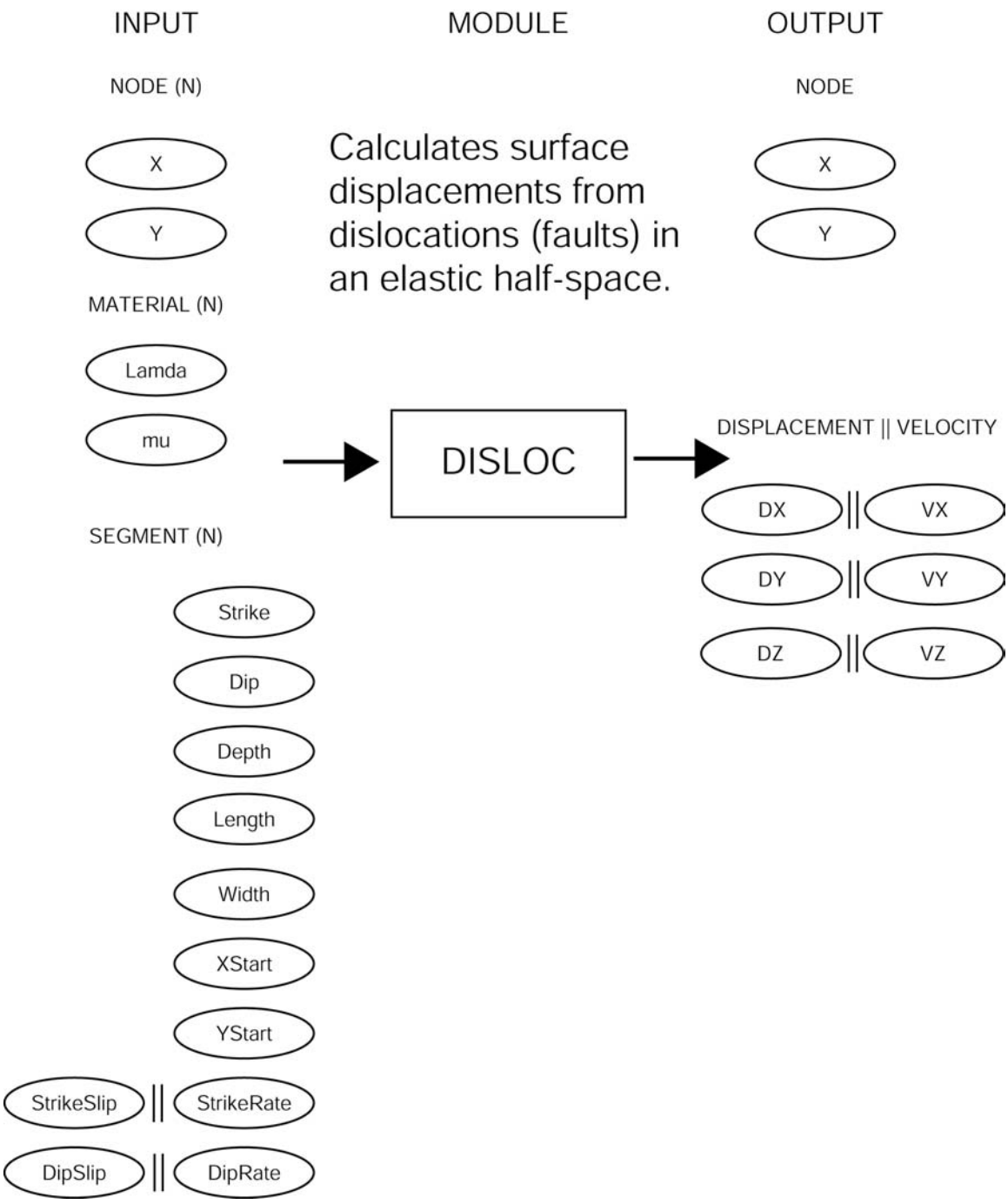

Figure 5. Disloc program input and output parameters.

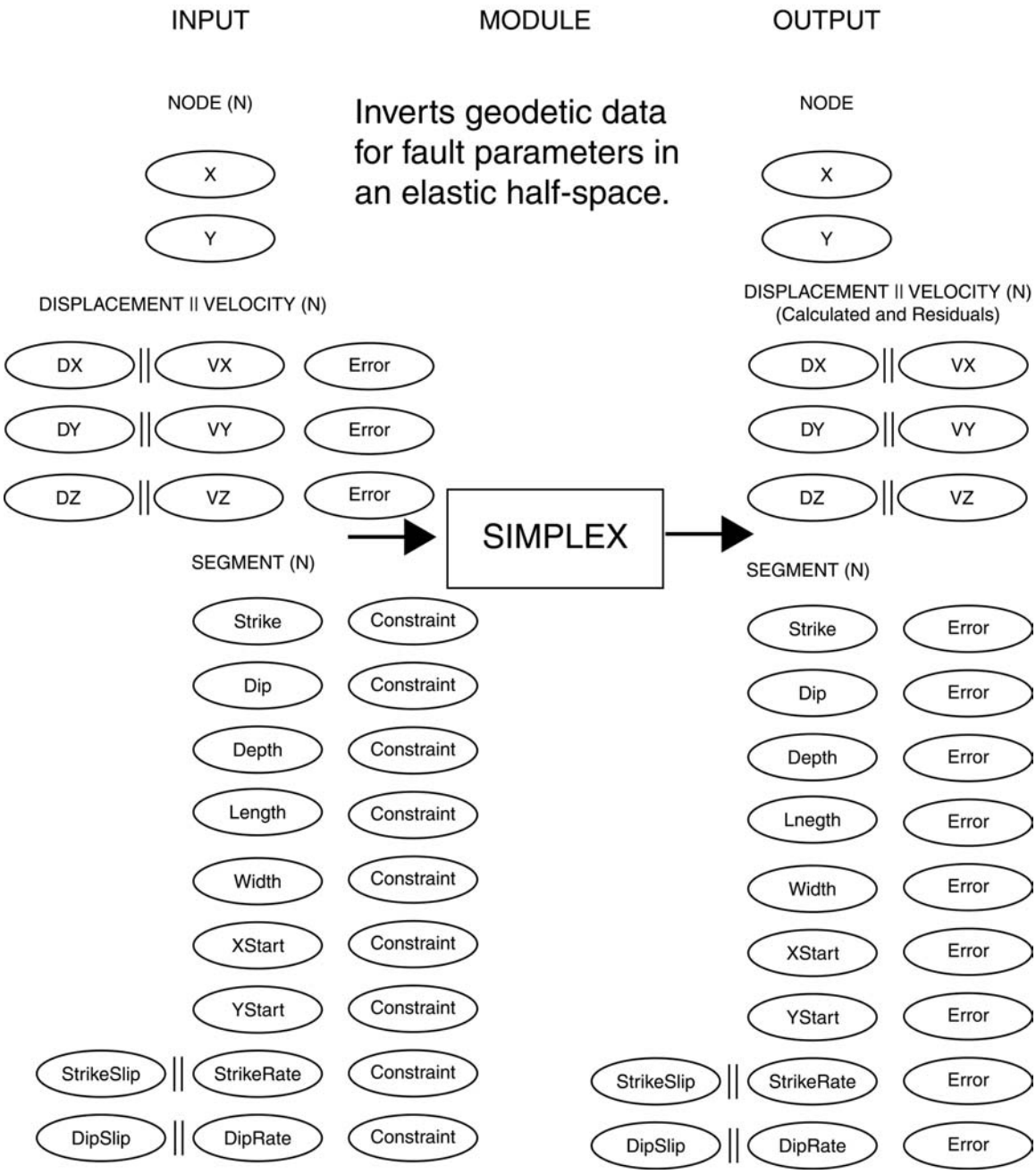

Figure 6. Simplex program input and output parameters.

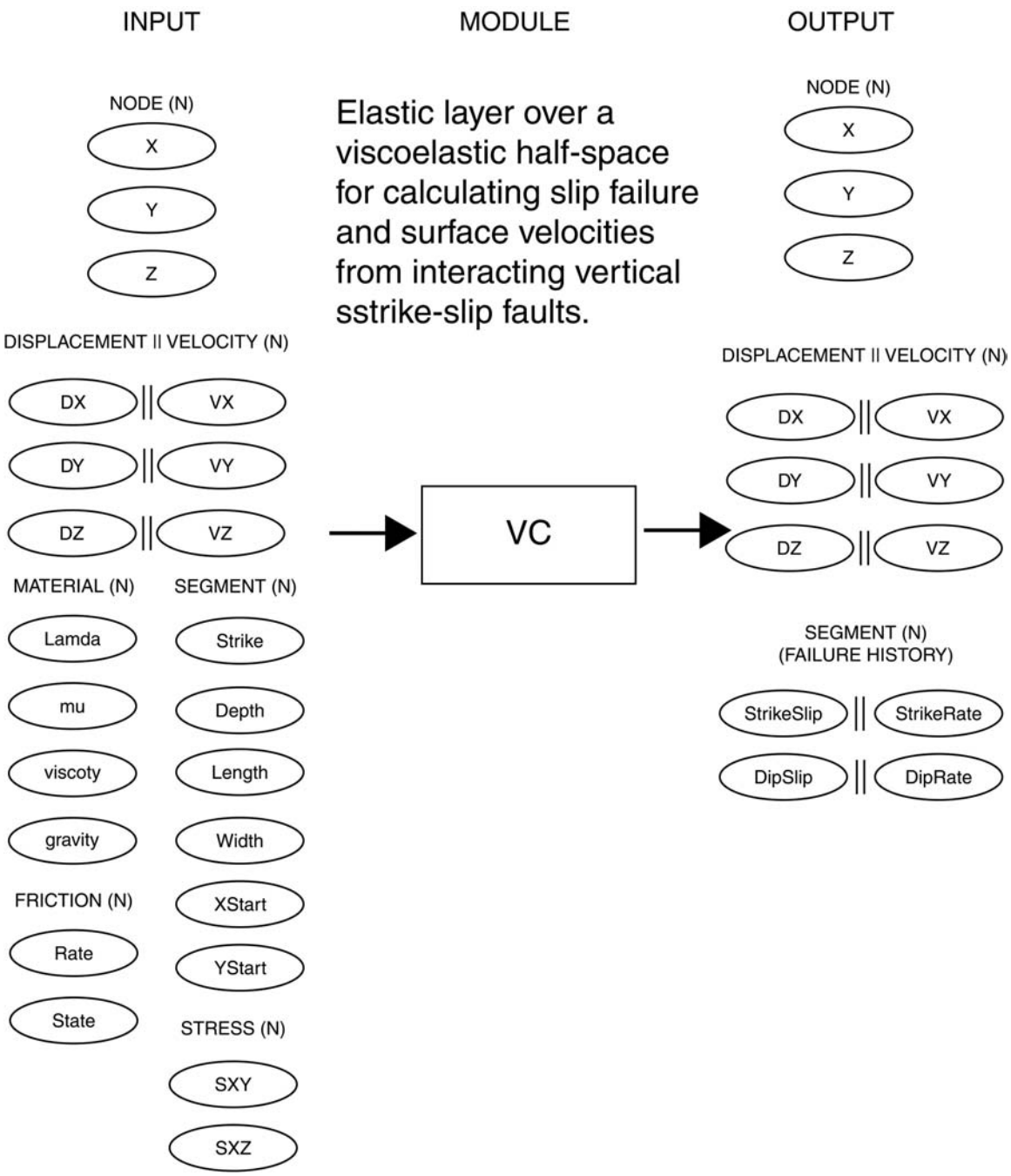

Figure 7. Virtual California program input and output parameters.

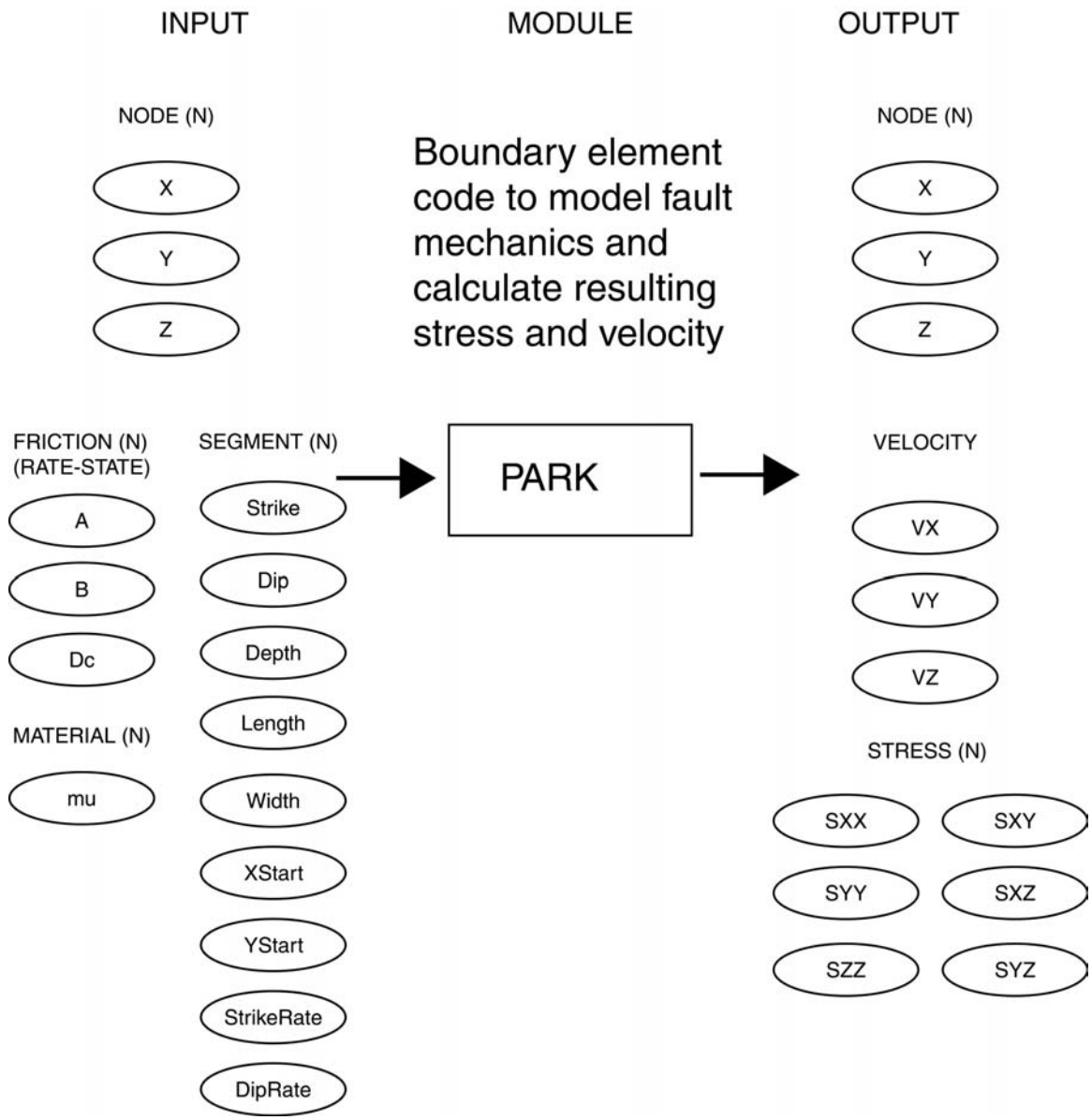

Figure 8. Park program input and output parameters.

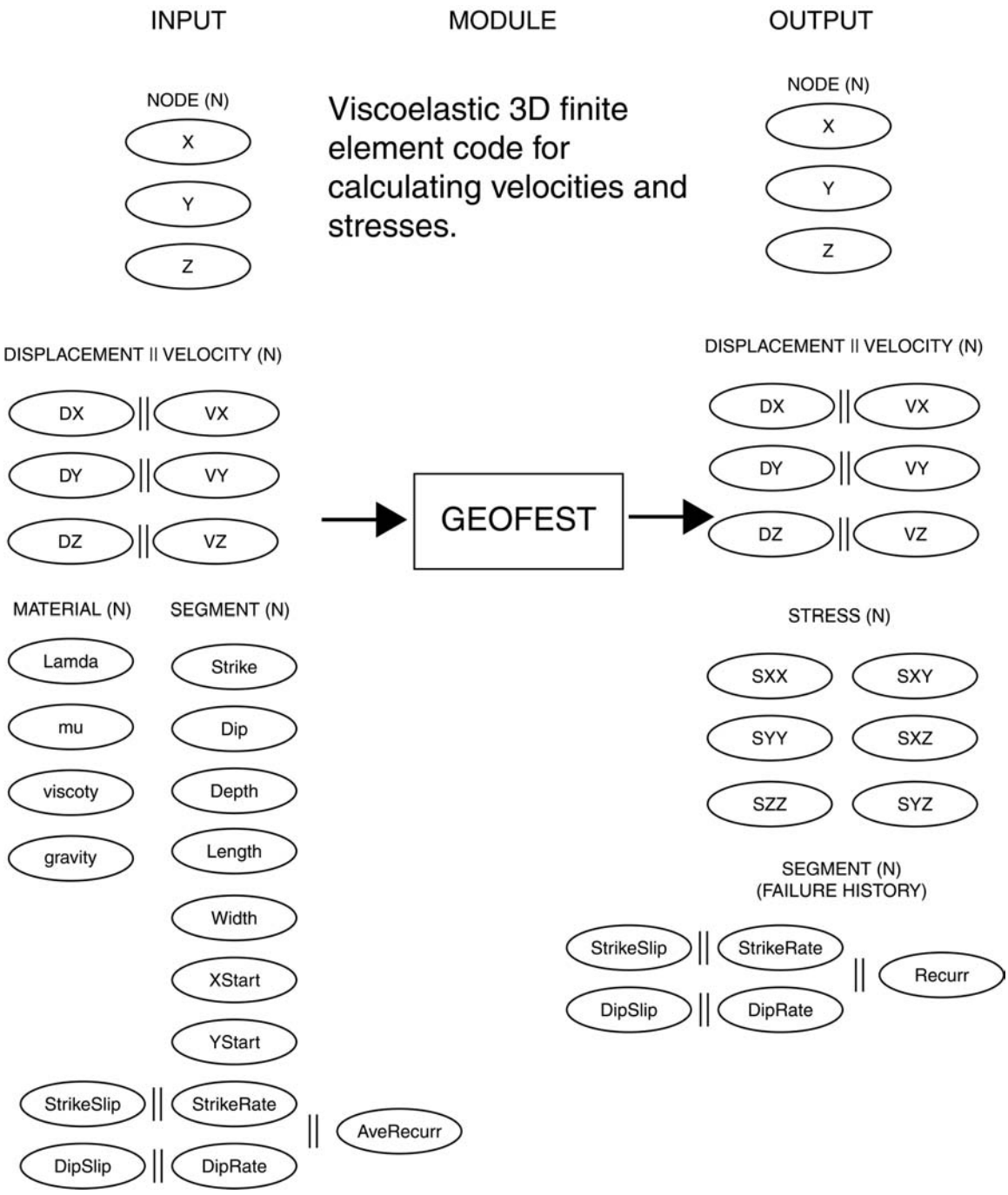

Figure 8. GeoFEST program input and output parameters.

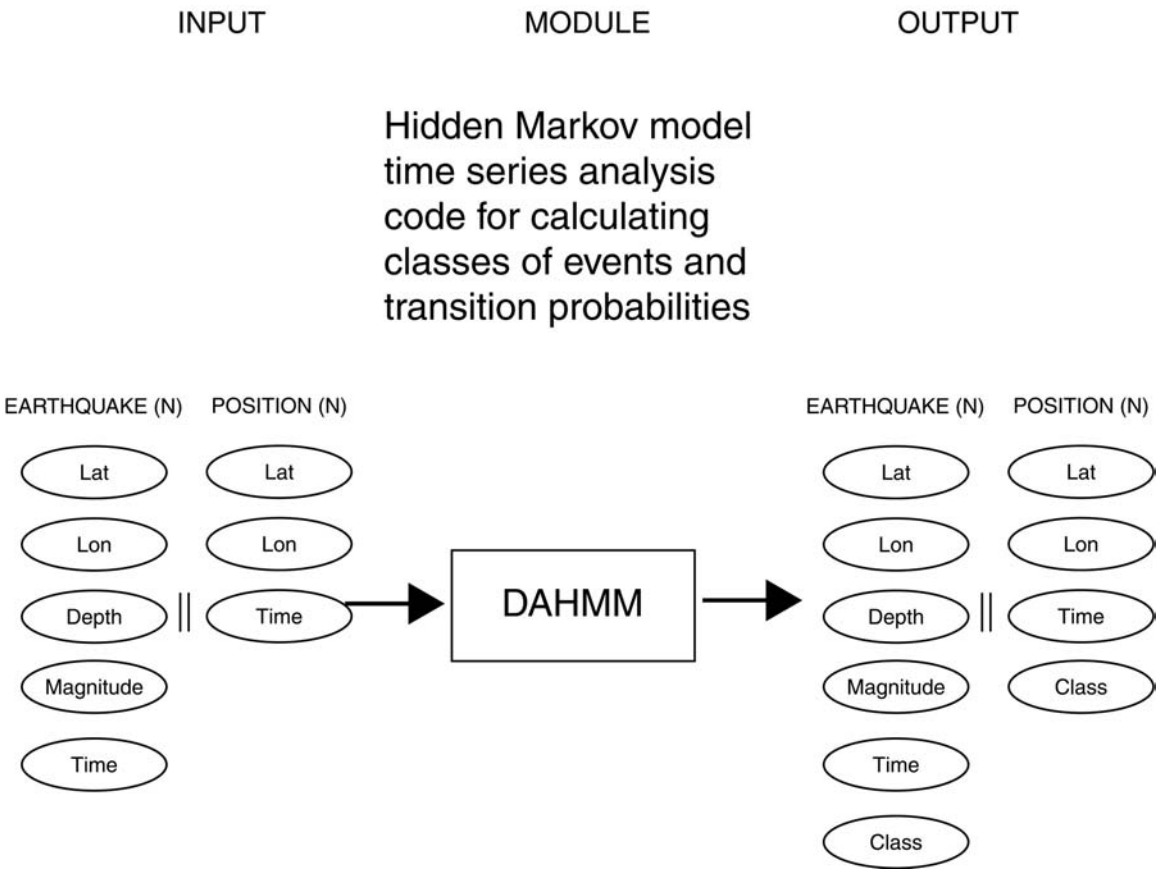

Figure 9. DAHMM program input and output parameters.

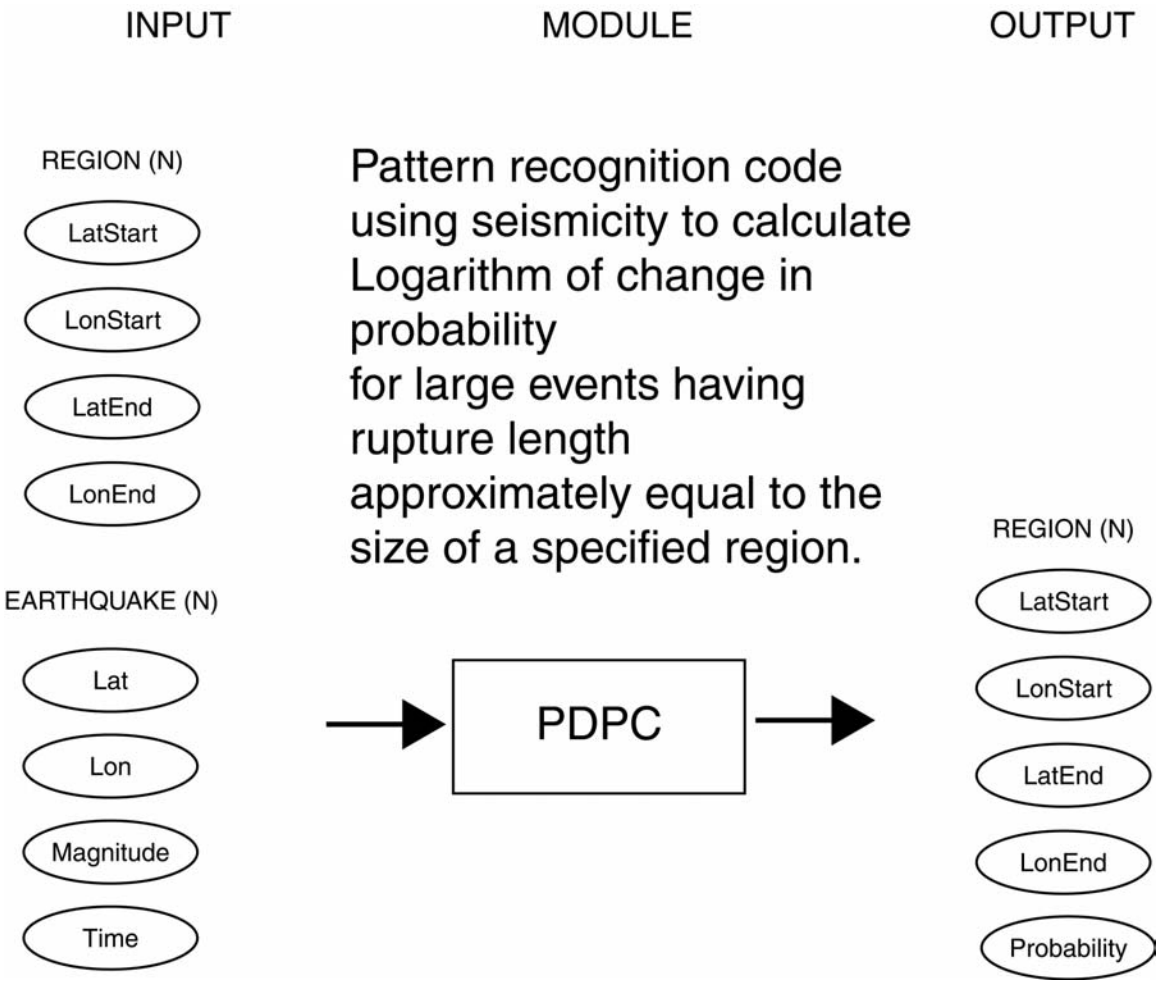

Figure 10. PDPC program input and output parameters.